Coding Style Guidelines

Use Code Change Helper to submit code changes.

Clojure Coding Style Guidelines

General Guidelines

- Basic Clojure Style as documented here: https://github.com/bbatsov/clojure-style-guide
- Code should be properly indented. Use the lisp indent plugin in Sublime to help.
    - Select all then command + i
- Almost all functions should have a doc string.
- All namespaces should have a doc string.
    - An exception to this rule is test namespaces which are usually obvious a mirror for testing the implementation namespace.
- Commented out code should be removed.
    - One exception is example code in a comment block.
- Insert newlines in long lines of code.

Pure Functions vs Functions with Side Effects

We should favor pure functions (functions with no side effects) vs functions with side effects. They are easier to test, reason about, parallelize, etc. We should put "pure" metadata on the function vars to indicate that they are pure. The metadata is mostly for readability but we may be able to build logic in the future based on that information. Pure functions can be tested easily with unit tests reducing the need for as many integration tests which are slower.

Example Pure Function:

```
(defn ^:pure foo
"I have no side effects"
[x]
(+ x 1))
```

When possible (and reasonable) we should extract a pure function from a side effecting one. Depending on the functionality extracted there may not be a good name for the pure one. If it encapsulates most of the logic of the stateful function we can use an asterisk in the name of the pure function.

```
(defn- ^:pure foo*
"I have no side effects"
[x]
(+ x 1))

(defn foo
"I launch rockets with some logic"
[x]
(launch-rocket (foo* x)))
```

Comments

Numbers of semicolons

- Use two semicolons for a comment on it's own line.
- Use one semicolon for a comment at the end of a line.
- Use a single space as a separator for sentences.

Destructuring

Don't use destructuring in the parameters to a public function of a namespace.

It becomes hard to understand the arguments. Use a let with destructuring in those cases within the function. You can use destructuring within

private functions of a namespace.

Bad:

```
(defn my-public-fn [{:keys [db cache}]
  ;;; stuff happens here)
```

Good:

```
(defn my-public-fn [
  (let [{:keys [db cache} system]
    ;;; stuff happens here))
```

Validations

Separate validations into two functions. One which does the validation and returns a list of errors and another which calls that function and throws the error.

This lets us easily test the validation code.

Example:

```
(defn id-set-validation [thing]
(when-not (:id thing)
["Id is required."]))
```

```
(defn validate-thing
[thing]
(when-let [errors (id-set-validation thing)]
(errors/throw-service-errors :invalid-data [errors])))
```

Testing

Put expected values first in assertions.

Sometimes the is calls can be very long. Putting it in the same order every time aids readability and understanding.

Bad:

```
(is (= x 5))
```

Good:

```
(is (= 5 x))
```

Avoid Using with-redefs

The with-redefs function is sometimes used for things like mocking in Clojure. It is not thread safe which can lead to problems since we run our tests in parallel. See http://dev.clojure.org/jira/browse/CLJ-1104

Namespaces

Follow the guidelines listed here https://stuartsierra.com/2016/clojure-how-to-ns.html in addition to rules below.

Rules for requiring

- Don't use "use". Use require :refer with a specific list. refer :all allowed in some special cases like tests.
- Remove unused references in the :require block of a namespace declaration.
- Require things with consistent aliases. Some common ones listed below
    - clojure.string :as str
    - clojure.set :as set
    - clojure.java.io :as io

Requiring for indirect usage

Sometimes we have to require things but not directly use them. They're only required so that the multimethods or protocol implementations are provided. These items should be required in their own :require block below the main require block with a single comment explaining the purpose. They should not be wrapped in brackets or parentheses. They should not be required with an alias.